

(19) 日本国特許庁 (JP)

(12) 特 許 公 報 (B2)

(11) 特許番号

特許第4784792号  
(P4784792)

(45) 発行日 平成23年10月5日 (2011. 10. 5)

(24) 登録日 平成23年7月22日 (2011. 7. 22)

(51) Int. Cl.

F I

G O 6 F 9/50 (2006. 01)

G O 6 F 9/46 4 6 5 E

G O 6 F 9/45 (2006. 01)

G O 6 F 9/44 3 2 2 H

G O 6 F 12/08 (2006. 01)

G O 6 F 12/08 5 0 9 F

G O 6 F 12/08 5 2 3 Z

G O 6 F 12/08 5 5 1 C

請求項の数 5 (全 12 頁)

(21) 出願番号 特願平11-363702

(22) 出願日 平成11年12月22日 (1999. 12. 22)

(65) 公開番号 特開2001-175619 (P2001-175619A)

(43) 公開日 平成13年6月29日 (2001. 6. 29)

審査請求日 平成18年4月25日 (2006. 4. 25)

審判番号 不服2009-12260 (P2009-12260/J1)

審判請求日 平成21年7月6日 (2009. 7. 6)

(73) 特許権者 899000068

学校法人早稲田大学

東京都新宿区戸塚町 1 丁目 1 0 4 番地

(74) 代理人 100075513

弁理士 後藤 政喜

(74) 代理人 100114236

弁理士 藤井 正弘

(72) 発明者 笠原 博徳

東京都新宿区大久保 3-4-1 早稲田大

学理工学部電気電子情報工学科

(72) 発明者 木村 啓二

東京都新宿区大久保 3-4-1 早稲田大

学理工学部電気電子情報工学科

最終頁に続く

(54) 【発明の名称】 マルチプロセッサ

(57) 【特許請求の範囲】

【請求項 1】

C P U と、前記 C P U に接続されているネットワークインタフェースと、コンパイラによりスタティックスケジューリングされたプログラムの実行時に転送されるデータを格納し、他のプロセッシングエレメントからアクセス可能な分散共有メモリと、当該プロセッシングエレメントだけからアクセス可能なローカルデータメモリと、を備える複数のプロセッシングエレメントと、

前記各プロセッシングエレメントに接続され、前記各プロセッシングエレメントによって共有され、コンパイラによりダイナミックスケジューリングされたプログラムの実行時に使用されるデータを格納する集中共有メモリと、を備えるマルチプロセッサであって、

前記集中共有メモリは、前記各プロセッシングエレメントと同じチップに設けられたオンチップの第 1 集中共有メモリと、前記いずれのプロセッシングエレメントとも異なるチップに設けられたオフチップの第 2 集中共有メモリとを含み、

前記分散共有メモリは、スタティックスケジューリングされたプログラムの実行時に、プロセッシングエレメント間のデータ転送に使用され、

前記ローカルデータメモリは、当該プロセッシングエレメントに割り当てられたタスクにおいて使用されるローカルデータを保持するために使用され、

前記各プロセッシングエレメントに割り当てられたタスク間で共通に使用されるデータが、前記各タスクで必要とされるとき以前に、データの消費先の前記プロセッシングエレメントの前記分散共有メモリへ転送され、

前記集中共有メモリは、粗粒度並列処理において条件分岐に対応するために使用されるダイナミックスケジューリングにおいて、プログラムの実行時までどのCPUにより使用されるかが決まっていなかったデータを格納することを特徴とするマルチプロセッサ。

【請求項2】

前記マルチプロセッサは、前記分散共有メモリに接続されるデータ転送コントローラを備え、

前記データ転送コントローラは、前記ローカルデータメモリから転送指示を読み出し、前記分散共有メモリからデータを読み出して、前記データの消費先のプロセッシングエレメントの分散共有メモリへ転送することを特徴とする請求項1に記載のマルチプロセッサ。

10

【請求項3】

前記分散共有メモリは、デュアルポートメモリで構成され、

前記データ転送コントローラは、前記分散共有メモリの一つのポートに接続されることを特徴とする請求項2に記載のマルチプロセッサ。

【請求項4】

前記第1集中共有メモリは、粗粒度並列処理において条件分岐に対応するために使用され、ダイナミックスケジューリングされたプログラムの実行時にタスク間の共有データを格納する集中共有メモリであり、

前記第2集中共有メモリは、大容量の集中共有メモリであることを特徴とする請求項1から3のいずれか一つに記載のマルチプロセッサ。

20

【請求項5】

前記分散共有メモリには、送信側の前記CPUの指示によって転送されるデータ及び同期フラグが書き込まれることを特徴とする請求項1から4のいずれか一つに記載のマルチプロセッサ。

【発明の詳細な説明】

【0001】

【発明の属する技術分野】

本発明は、複数のCPUを単一のチップに納めたシングルチッププロセッサのアーキテクチャに関し、より具体的には、マルチグレインのコンパイラ協調型シングルチップマルチプロセッサアーキテクチャと、それらを接続した高性能マルチプロセッサシステムアーキテクチャとに関する。

30

【0002】

【従来の技術】

現在、日本のスーパーコンピュータメーカーは世界でもトップのハードウェア技術を有し、現時点でのピーク性能は、数TFLOPSを越え、21世紀初頭には数十TFLOPS以上のピーク性能を持つマシンが開発されると予想される。しかし、現在のスーパーコンピュータは、ピーク性能の向上とともにプログラムを実行したときの実効性能との差が大きくなっている、すなわち価格性能比が必ずしも優れているとはいえない状況になっている。また、使い勝手としても、ユーザは問題中の並列性を抽出し、HPF、MPI、PVMなどの拡張言語あるいはライブラリを用いハードウェアを効果的に使用できるようなプログラムを作成しなければならず、一般のユーザには使い方が難しい、あるいは使いこなせないという問題が生じている。さらに、これらにも起因して、世界の高性能コンピュータの市場を拡大できないということが大きな問題となっている。

40

【0003】

この価格性能比、使いやすさの問題を解決し、スーパーコンピュータの市場を拡大するためには、ユーザが使い慣れているフォートラン、C等の逐次型言語で書かれたプログラムを自動的に並列化する自動並列化コンパイラの開発が重要となる。

【0004】

特に、21世紀初頭の汎用並びに組み込み用マイクロプロセッサ、家庭用サーバからスーパーコンピュータに至るマルチプロセッサシステムの主要アーキテクチャの一つとなると

50

考えられるシングルチップマルチプロセッサについて検討を行うことは重要である。さらに、シングルチップマルチプロセッサについても、従来からある主記憶共有アーキテクチャでは十分な性能と優れた価格性能比は得られない。したがって、プログラム中の命令レベルの並列性、ループ並列性、粗粒度並列性をフルに使用できるマルチグレイン並列処理のように、真に実行すべき命令列からより多くの並列性を抽出し、システムの価格性能比を向上し、誰にでも使えるユーザフレンドリなシステムの構築を可能とする新しい自動並列化コンパイル技術と、それを生かせるようなアーキテクチャの開発が重要である。

【 0 0 0 5 】

【発明が解決しようとする課題】

したがって、本発明は、マルチグレイン並列化をサポートするコンパイラ協調型のシングルチップマルチプロセッサおよびそれを結合したハイパフォーマンスマルチプロセッサシステムを提供することを目的とする。

【 0 0 0 6 】

【課題を解決するための手段】

本発明は、CPUと、前記CPUに接続されているネットワークインタフェースと、コンパイラによりスタティックスケジューリングされたプログラムの実行時に転送されるデータを格納し、他の前記プロセッシングエレメントからアクセス可能な分散共有メモリと、当該プロセッシングエレメントだけからアクセス可能なローカルデータメモリと、を備える複数のプロセッシングエレメントと、前記各プロセッシングエレメントに接続され、前記各プロセッシングエレメントによって共有され、コンパイラによりダイナミックスケジューリングされたプログラムの実行時に使用されるデータを格納する集中共有メモリと、を備えるマルチプロセッサであって、前記集中共有メモリには、前記各プロセッシングエレメントと同じチップに設けられたオンチップの第1集中共有メモリと、前記いずれのプロセッシングエレメントとも異なるチップに設けられたオフチップの第2集中共有メモリとを含み、前記分散共有メモリは、スタティックスケジューリングされたプログラムの実行時に、プロセッシングエレメント間のデータ転送に使用され、前記ローカルデータメモリは、当該プロセッシングエレメントに割り当てられたタスクにおいて使用されるローカルデータを保持するために使用され、前記各プロセッシングエレメントに割り当てられたタスク間で共通に使用されるデータが、前記各タスクで必要とされるとき以前に、データの消費先の前記プロセッシングエレメントの前記分散共有メモリへ転送され、前記集中共有メモリは、粗粒度並列処理において条件分岐に対応するために使用されるダイナミックスケジューリングにおいて、プログラムの実行時までどのCPUにより使用されるかが決まっていなかったデータを格納することを特徴とするマルチプロセッサを提供する。

【 0 0 0 7 】

また、本発明は、前記マルチプロセッサは、前記分散共有メモリの一つのポートに接続されるデータ転送コントローラを備え、前記データ転送コントローラは、前記ローカルデータメモリから転送指示を読み出し、前記分散共有メモリからデータを読み出して、前記データの消費先のプロセッシングエレメントの分散共有メモリへ転送することを特徴とするマルチプロセッサを提供する。

【 0 0 0 8 】

【発明の実施の形態】

本発明はマルチグレイン並列化をサポートするシングルチップマルチプロセッサを提供する。本発明の一実施形態であるシングルチップマルチプロセッサのアーキテクチャを図1に示す。図1においては、複数のプロセッシングエレメント ( $PE_0, PE_1, \dots, PE_n$ ) を含んでなる複数 ( $m+1$  個) のシングルチップマルチプロセッサ ( $SCM_0, SCM_1, SCM_2, \dots, SCM_m, \dots$ ) 10 と、共有メモリのみからなる複数 ( $j+1$  個) の集中共有メモリチップ ( $CSM_0, \dots, CSM_j$ ) (ただし、CSMは要求されるシステム条件によっては1個もなくともよい) と、入出力制御を行う複数 ( $k+1$  個) のシングルチップマルチプロセッサで構成される入出力チップ ( $I/O\_SCM_0, \dots, I/O\_SCM_k$ ) (ただし、入出力制御に関しては既存技術のプロセッサを用

いることもできる)とが、チップ間接続ネットワーク12によって接続されている。このインタチップ接続ネットワーク12は、クロスバー、バス、マルチステージネットワークなど既存のネットワーク技術を利用して実現できるものである。

#### 【0009】

図1に示した形態においては、I/Oデバイスは要求される入出力機能に応じて $k+1$ 個のSCMで構成される入出力制御チップに接続している構成となっている。さらに、このチップ間接続ネットワーク12には、システム中の全プロセッシングエレメントにより共有されているメモリのみから構成される $j+1$ 個の集中共有メモリ(CSM: centralized shared memory)チップ14が接続されている。これは、SCM10内にある集中共有メモリを補完する働きをするものである。

10

#### 【0010】

マルチグレイン並列処理とは、サブルーチン、ループ、基本ブロック間の粗粒度並列性、ループタイプイタレーション間の中粒度並列性(ループ並列性)、ステートメントあるいは命令間の(近)細粒度並列性を階層的に利用する並列処理方式である。この方式により、従来の市販マルチプロセッサシステム用自動並列化コンパイラで用いられていたループ並列化、あるいはスーパースカラ、VLWにおける命令レベル並列化のような局所的で単一粒度の並列化とは異なり、プログラム全域にわたるグローバルかつ複数粒度によるフレキシブルな並列処理が可能となる。

#### 【0011】

[粗粒度タスク並列処理(マクロデータフロー処理)]

20

単一プログラム中のサブルーチン、ループ、基本ブロック間の並列性を利用する粗粒度並列処理は、マクロデータフロー処理とも呼ばれる。ソースとなる例えばフォートランプログラムを、粗粒度タスク(マクロタスク)として、繰り返しブロック(RB: repetition block)、サブルーチンブロック(SB: subroutine block)、疑似代入文ブロック(BPA: block of pseudo assignment statements)の3種類のマクロタスク(MT)に分解する。RBは、各階層での最も外側のナチュラルループであり、SBはサブルーチン、BPAはスケジューリングオーバーヘッドあるいは並列性を考慮し融合あるいは分割された基本ブロックである。ここで、BPAは、基本的には通常の基本ブロックであるが、並列性抽出のために単一の基本ブロックを複数に分割したり、逆に一つのBPAの処理時間が短く、ダイナミックスケジューリング時のオーバーヘッドが無視できない場合には、複数のBPAを融合し得一つのBPAを生成する。最外側ループであるRBがDoallループであるときは、ループインデックスを分割することにより複数の部分Doallループに分割し、分割後の部分Doallループを新たにRBと定義する。また、サブルーチンSBは、可能な限りインライン展開するが、コード長を考慮し効果的にインライン展開ができないサブルーチンはそのままSBとして定義する。さらに、SBやDoall不可能なRBの場合、これらの内部の並列性に対し、階層的マクロデータフロー処理を適用する。

30

#### 【0012】

次に、マクロタスク間の制御フローとデータ依存を解析し、図2のようなマクロフローグラフ(MFG)を生成する。MFGでは、各ノードがマクロタスク(MT)、点線のエッジが制御フロー、実線のエッジがデータ依存、ノード内の小円が条件分岐文を表している。また、MT7のループ(RB)は、内部で階層的にMTおよびMFGを定義できることを示している。

40

#### 【0013】

次に、マクロタスク間制御依存およびデータ依存より各マクロタスクが最も早く実行できる条件(最早実行可能条件)すなわちマクロタスク間の並列性を検出する。この並列性をグラフ表現したのが図3に示すマクロタスクグラフ(MTG)である。MTGでも、ノードはMT、実線のエッジがデータ依存、ノード内の小円が条件分岐文を表す。ただし、点線のエッジは拡張された制御依存を表し、矢印のついたエッジは元のMFGにおける分岐先、実線の円弧はAND関係、点線の円弧はOR関係を表している。例えば、MT6へのエッジは、MT2中の条件分岐がMT4の方向に分岐するか、MT3の実行が終了したと

50

き、MT6が最も早く実行が可能になることを示している。

【0014】

そして、コンパイラは、MTG上のMTをプロセッサクラスタ（コンパイラあるいはユーザによりソフトウェア的に実現されるプロセッサのグループ）へコンパイル時に割り当てを行う（スタティックスケジューリング）か、実行時に割り当てを行うためのダイナミックスケジューリングコードを、ダイナミックCPアルゴリズムを用いて生成し、これをプログラム中に埋め込む。これは、従来のマルチプロセッサのようにOSあるいはライブラリに粗粒度タスクの生成、スケジューリングを依頼すると、数千から数万クロックのオーバーヘッドが生じてしまう可能性があり、それを避けるためである。このダイナミックなスケジューリング時には、実行時までどのプロセッサでタスクが実行されるか分からないため、タスク間共有データは全プロセッサから等距離に見える集中共有メモリに割り当てられる。

10

【0015】

また、このスタティックスケジューリングおよびダイナミックスケジューリングコードの生成の時には、各プロセッサ上のローカルメモリあるいは分散共有メモリを有効に使用し、プロセッサ間のデータ転送量を最小化するためのデータローカライゼーション手法も用いられる。

【0016】

データローカライゼーションは、MTG上でデータ依存のある複数の異なるループにわたリイタレーション間のデータ依存を解析し（インターループデータ依存解析）、データ転送が最小になるようにループとデータを分割（ループ整合分割）後、それらのループとデータが同一のプロセッサにスケジューリングされるように、コンパイル時にそれらのループを融合するタスク融合方式か、実行時に同一プロセッサへ割り当てられるようにコンパイラが指定するパーシャルスタティックスケジューリングアルゴリズムを用いてダイナミックスケジューリングコードを生成する。このデータローカライゼーション機能を用いて各ローカルメモリの有効利用を行うことができる。

20

【0017】

またこの際、データローカライゼーションによっても除去できなかったプロセッサ間のデータ転送を、データ転送とマクロタスク処理をオーバーラップして行うことにより、データ転送オーバーヘッドを隠蔽しようとするプレロード・ポストストアスケジューリングアルゴリズムも使用される。このスケジューリングの結果に基づいて各プロセッサ上のデータ転送コントローラを利用したデータ転送が実現される。

30

【0018】

[ループ並列処理（中粒度並列処理）]

マルチグレイン並列化では、マクロデータフロー処理によりプロセッサクラスタ（PC）に割り当てられるループ（RB）は、そのRBがDoallあるいはDoacrossループの場合、PC内のプロセッシングエレメント（PE）に対してイタレーションレベルで並列化処理（分割）される。

【0019】

ループストラクチャリングとしては、以下のような従来の技術をそのまま利用できる。

40

- （a）ステートメントの実行順序の変更
- （b）ループディストリビューション
- （c）ノードスプリッティングスカラエクスパンション
- （d）ループインターチェンジ
- （e）ループアンローリング
- （f）ストリップマイニング
- （g）アレイプライベートイゼーション
- （h）ユニモジュラー変換（ループリバーサル、パーミュテーション、スキューイング）

【0020】

また、ループ並列化処理が適用できないループに関しては、図4のようにループボディ部

50

を次に述べる（近）細粒度並列処理か、ボディ部を階層的にマクロタスクに分割しマクロデータフロー処理（粗粒度タスク並列処理）を適用する。

#### 【0021】

##### 〔（近）細粒度並列処理〕

PCに割り当てられるMTがBPAまたはループ並列化或いは階層的にマクロデータフロー処理を適用できないRB等の場合には、BPA内部のステートメント或いは命令を近細粒度タスクとしてPC内プロセッサで並列処理する。

#### 【0022】

マルチプロセッサシステム或いはシングルチップマルチプロセッサ上での近細粒度並列処理では、プロセッサ間の負荷バランスだけでなくプロセッサ間データ転送をも最少にするようにタスクをプロセッサにスケジューリングしなければ、効率よい並列処理は実現できない。さらに、この近細粒度並列処理で要求されるスケジューリングでは、図4のタスクグラフに示すように、タスク間にはデータ依存による実行順序の制約があるため強NP完全な非常に難しいスケジューリング問題となる。このグラフは、無サイクル有向グラフである。図中、各タスクは各ノードに対応している。ノード内の数字はタスク番号 $i$ を表し、ノードの脇の数字はプロセッシングエレメント上でのタスク処理時間 $t_i$ を表す。また、ノード $N_i$ から $N_j$ に向けて引かれたエッジは、タスク $T_i$ が $T_j$ に先行するという半順序制約を表している。タスク間のデータ転送時間も考慮する場合、各々のエッジは一般に可変な重みを持つ。タスク $T_i$ と $T_j$ が異なるプロセッシングエレメントへ割り当てられた場合、この重み $t_{ij}$ がデータ転送時間となる。図4においては、データ転送および同期に要する時間を9クロックと仮定している。逆にこれらのタスクが同一プロセッシングエレメントに割り当てられた場合、重み $t_{ij}$ は0となる。

#### 【0023】

このようにして生成されたタスクグラフを各プロセッサにスタティックにスケジューリングする。この際、スケジューリングアルゴリズムとして、データ転送オーバーヘッドを考慮し実行時間を最小化するヒューリスティックアルゴリズム、例えばCP/DT/MISF法、CP/ETF/MISF法、ETF/CP法、あるいはDT/CP法の4手法を自動的に適用し最良のスケジュールを選ぶことができる。また、このようにタスクをスタティックにプロセッサに割り当てることにより、BPA内で用いられるデータのローカルメモリ、分散共有メモリ、レジスタへの配置等、データのメモリへの最適化やデータ転送・同期オーバーヘッドの最小化といった各種の最適化が可能になる。

#### 【0024】

スケジューリング後、コンパイラはプロセッシングエレメントに割り当てられたタスクの命令列を順番に並べ、データ転送命令や同期命令を必要な箇所に挿入することにより、各プロセッサ用のマシンコードを生成する。近細粒度タスク間の同期にはバージョンナンバー法を用い、同期フラグの受信は受信側プロセッシングエレメントのビジーウェイトによって行われる。ここで、データ転送および同期フラグのセットは、送信側のプロセッサが受信側のプロセッサ上の分散共有メモリに直接書き込むことにより低オーバーヘッドで行うことができる。

#### 【0025】

マシンコード生成時、コンパイラはスタティックスケジューリングの情報を用いたコード最適化を行うことができる。例えば、同一データを使用する異なるタスクが同一プロセッシングエレメントに割り当てられたとき、レジスタを介してそのデータを受け渡しすることができる。また、同期のオーバーヘッドを最小化するため、タスクの割り当て状況や実行順序から、冗長な同期を除去することもできる。特に、シングルチップマルチプロセッサでは、コード生成時に厳密なコード実行スケジューリングを行うことにより、実行時のデータ転送タイミングを含めたすべての命令実行をコンパイラが制御し、すべての同期コードを除去して並列実行を可能とする無同期並列化のような究極的な最適化も行える。

#### 【0026】

上述のようなマルチグレイン並列処理をマルチプロセッサシステム上で実現するため、一

10

20

30

40

50

例として、シングルチップマルチプロセッサ（ＳＣＭ）１０は図１に示すようなアーキテクチャを有する。

【 0 0 2 7 】

図 1 において示したアーキテクチャにおいては、CPU 20に加えて、分散共有メモリ(DSM: distributed shared memory) 22とアジャスタブルプリフェッチ命令キャッシュ 24が各SCM 10に設けられている。ここで用いられるCPU 20は、特に限定されず、整数演算や浮動小数点演算が可能なものであればよい。例えば、ロード/ストアアーキテクチャのシンプルなシングルイシューRISCアーキテクチャのCPUを用いることができるほか、スーパースカラプロセッサ、VLIWプロセッサなども用いることができる。分散共有メモリ 22は、デュアルポートメモリで構成されており、他のプロセッシングエレメントからも直接リード/ライトができるようになっており、上に説明した近細粒度タスク間のデータ転送に使用する。

【 0 0 2 8 】

アジャスタブルプリフェッチ命令キャッシュ 24 は、コンパイラあるいはユーザからの指示で、将来実行すべき命令をメモリあるいは低レベルキャッシュからプリフェッチするものである。このアジャスタブルプリフェッチ命令キャッシュ 24 は、複数ウェイのセットアソシアティブキャッシュにおいて、コンパイラ等のソフトから指示される、あるいはハードにより事前に決められたウェイに、将来実行されるライン（命令列）をフェッチできるようにするものである。その際、フェッチの単位としては、複数ラインの連続転送指示も行える。アジャスタブルプリフェッチ命令キャッシュ 24 は、命令キャッシュへのミスヒットを最小化させ、命令実行の高速化を可能にするコンパイラによる調整および制御を可能にするキャッシュシステムである。

【 0 0 2 9 】

すなわち、このアジャスタブルプリフェッチ命令キャッシュ24は、すべてのプログラム（命令列）がメモリサイズより小さいことを仮定しているローカルプログラムメモリとは異なり、大きなプログラムにも対応することができ、プログラムの特徴に応じ、プリフェッチをしない通常のキャッシュとしても使用できるし、逆にすべてコンパイラ制御によるプリフェッチキャッシュとして使え、ミスヒットのない（ノーミスヒット）キャッシュとして使用できるものである。

【 0 0 3 0 】

このようなアジャスタブルプリフェッチ命令キャッシュの構造の一例を図 5 に示す。図 5 に示された  $n$  ウェイのセットアソシエティブキャッシュにおいては、コンパイラあるいはユーザがプログラムに応じて指定する  $j$  ウェイをプリフェッチ（事前読み出し）するエリアとして使用できるものである。コンパイラにより挿入されたプリフェッチ命令（ラインごとではなく複数ラインのプリフェッチも可能）により、命令実行の前に必要な命令が命令キャッシュ上に存在することを可能とし、高速化が実現できる。プロセッシングエレメントは、 $n$  ウェイすべてを通常のキャッシュと同様に読み出すことができる。ラインのリプレースは通常の LRU（least recently used）法で行われる。そして、各セット（集合）中のウェイには、通常、自由に転送されたラインを格納できるが、プリフェッチ用に指定されたウェイにはプリフェッチ命令によって CSM から転送されたラインのみ格納される。それ以外のウェイは通常のキャッシュと同様にラインを割り当てられる。プリフェッチキャッシュコントローラは、コンパイラからの指示により、命令を CSM からプリフェッチする。このときの転送の単位は、1 ラインから複数ラインである。コンパイラが  $j$  ウェイ分のプリフェッチエリアを指定し、それ以外の  $(n - j)$  ウェイ分のエリアは通常のキャッシュとして使用される。

【 0 0 3 1 】

さらに、図 1 のアーキテクチャにおいては、ローカルデータメモリ（LDM）26 が設けられている。このローカルデータメモリ 26 は、各プロセッシングエレメント 16 内だけでアクセスできるメモリであり、データローカライゼーション技術などにより、各プロセッシングエレメント 16 に割り当てられたタスク間で使用されるローカルデータを保持す

るために使用される。また、このローカルデータメモリ 26 は、対象とするアプリケーションプログラムに対しコンパイラあるいはユーザがデータのローカルメモリへの分割配置が可能な場合には、ローカルメモリとして使用され、ローカルメモリを有効に使用できない場合には、レベル 1 キャッシュ (D キャッシュ) に切り替えて使用できるようにすることが好ましい。また、ゲーム機等のリアルタイム応用に専ら用いられるような場合には、ローカルメモリだけとして設計することも可能である。基本的に各プロセッシングエレメント内で使用されるメモリであるため、共有メモリに比べチップ面積を消費しないので、相対的に大きな容量をとれるものである。

#### 【0032】

粗粒度並列処理では、条件分岐に対処するためにダイナミックスケジューリングが使用される。この場合、マクロタスクがどのプロセッサで実行されるかは、コンパイル時には分からない。したがって、ダイナミックにスケジューリングされるマクロタスク間の共有データは、集中共有メモリ (CSM: centralized shared memory) に配置できることが好ましい。そのため、本実施形態においては、各プロセッシングエレメント 16 が共有するデータを格納する集中共有メモリ 28 を各 SCM 内に設けるほか、さらに、チップ間接続ネットワーク 12 につながれた集中共有メモリ 14 を設けている。このチップ内の集中共有メモリ 28 は、チップ 10 内のすべてのプロセッシングエレメント 16 から、そして複数チップの構成では他のチップ上のプロセッシングエレメントからも共有されるデータを保存するメモリである。チップ外の集中共有メモリ 14 も同様に各プロセッシングエレメントにより共有されるメモリである。したがって、実際の設計上、集中共有メモリ 28、14 は、物理的に各チップに分散されているが、論理的にはどのプロセッシングエレメントからも等しく共有することができるものである。すべてのプロセッシングエレメントから等距離に見えるようにインプリメントすることもできるし、自チップ内のプロセッシングエレメントからは近く見えるようにインプリメントすることも可能である。

#### 【0033】

単一の SCM チップからなるシステムでは、チップ内のプロセッシングエレメント (PE) 16 間で共有される等距離の共有メモリとしてこの集中共有メモリ 28 を用いることができる。また、コンパイラの最適化が困難である場合には、L2 キャッシュとして使用することができる。このメモリ 28、14 には、ダイナミックタスクスケジューリング時にタスク間で共有されるデータを主に格納する。また、別のチップとなった集中共有メモリ 14 は、SCM チップ 10 内の集中共有メモリ 28 の容量が足りない場合、必要に応じて、メモリのみからなる大容量集中共有メモリチップを任意の数接続することができる。

#### 【0034】

また、粒度によらずスタティックスケジューリングが適用できる場合には、あるマクロタスクが定義する共有データをどのプロセッサが必要とするかはコンパイル時に分かるため、生産側のプロセッサが消費側のプロセッサの分散共有メモリにデータと同期用のフラグを直接書き込めることが好ましい。

#### 【0035】

データ転送コントローラ (DTC) 30 は、コンパイラあるいはユーザの指示により自プロセッシングエレメント上の DSM 22 や、自あるいは他の SCM 10 内の CSM 28、あるいは他のプロセッシングエレメント上の DSM との間でデータ転送を行う。複数の SCM からなる構成を採用する場合には、他の SCM 上の CSM や DSM との間でのデータ転送、あるいは、独立した CSM との間でのデータ転送を行う。

#### 【0036】

図 1 におけるローカルデータメモリ 26 とデータ転送コントローラ 30 との間の点線は、用途に応じて、データ転送コントローラ 30 がローカルデータメモリ (D キャッシュ) 26 にアクセスできる構成をとってもよいことを表している。このような場合、ローカルデータメモリ 26 を介して CPU 20 が転送指示をデータ転送コントローラ 30 に与えたり、転送終了のチェックを行う構成をとることができる。

#### 【0037】

10

20

30

40

50



データ転送コントローラ 30 へのデータ転送の指示は、ローカルデータメモリ 26、DSM 22、あるいは専用のバッファ（図示しない）を介して行い、データ転送コントローラ 30 から CPU 20 へのデータ転送終了の報告は、ローカルメモリ、DSM あるいは専用のバッファを介して行う。このとき、どれを使うかはプロセッサの用途に応じプロセッサ設計時に決めるかあるいはハード的に複数の方法を用意し、プログラムの特性に応じコンパイラあるいはユーザがソフト的に使い分けられるようにする。

#### 【0038】

データ転送コントローラ 30 へのデータ転送指示（例えば何番地から内バイトのデータをどこにストアし、またロードするか、データ転送のモード（連続データ転送、ストライド、ストライド・ストライド転送など）など）は、コンパイラが、データ転送命令をメモリあるいは専用バッファに格納しておき、実行時にはどのデータ転送命令を実行するかの指示のみを出すようにして、データ転送コントローラ 20 の駆動のためのオーバヘッドを削減することが好ましい。

10

#### 【0039】

各 SCM チップ 10 内のプロセッシングエレメント 16 の間の接続は、各プロセッシングエレメントに設けられたネットワークインタフェース 32 を介して、チップ内接続ネットワーク（マルチバス、クロスバーなどからなる）34 によって達成されており、このチップ内接続ネットワーク 34 を介して、プロセッシングエレメントが共通の集中共有メモリ 28 に接続される。集中共有メモリ 28 は、チップの外にあるチップ間接続ネットワーク 12 に接続している。このチップ間接続ネットワークは、クロスバーネットワークあるいはバス（複数バスも含む）が特に好ましいが、多段結合網等でもかまわず、予算、SCM の数、アプリケーションの特性に応じて選ぶことができるものである。また、このチップ内接続ネットワーク 34 を介さずに、外部のチップ間接続ネットワーク 12 とネットワークインタフェース 32 を接続することも可能であり、このような構成は、システム中の全プロセッシングエレメントが平等に各チップ上に分散された集中共有メモリ、分散共有メモリにアクセスすることを可能にするほか、チップ間でのデータ転送が多い場合には、この直結バスを設けることにより、システム全体のデータ転送能力を大幅に高めることができる。

20

#### 【0040】

グローバルレジスタファイル 36 は、マルチポートレジスタであり、チップ内のプロセッシングエレメントにより共有されるレジスタである。たとえば、近細粒度タスク（分散共有メモリを用いた場合など）のデータ転送および同期に使用することができる。このグローバルレジスタファイルは、プロセッサの用途に応じて、省略することも可能なものである。

30

#### 【0041】

図 1 において、点線は、通信線を必要に応じて用意できることを意味しており、コストあるいはピン数などを考えて不必要あるいは困難な場合には、点線の接続はなくても動作することを示すものである。

#### 【0042】

以上のように、特定の実施の形態に基づいて本発明を説明してきたが、本発明の技術的範囲はこのような実施の形態に限定されるものではなく、当業者にとって容易な種々の変形を含むものである。

40

#### 【0043】

##### 【発明の効果】

上述のように、本発明のシングルチップマルチプロセッサによれば、価格性能比を改善し、高まりつつある半導体集積度にスケラブルな性能向上が可能である。また、本発明は、このようなシングルチップマルチプロセッサを複数含むシステムをも提供するが、そのようなシステムは、より一層の高速処理を可能にするものである。

##### 【図面の簡単な説明】

【図 1】本発明の 1 実施形態であるマルチグレイン並列処理用システムを示すブロックダ

50

イアグラムである。

【図2】本発明において用いることができるコンパイラにおける粗粒度並列処理のためのマクロフローグラフの一例を示すグラフである。

【図3】本発明において用いることができるコンパイラにおける粗粒度並列処理のためのマクロタスクグラフの一例を示すグラフである。

【図4】本発明において用いることができるコンパイラにおける近細粒度並列処理のための近細粒度タスクグラフの一例を示すグラフである。

【図5】本発明において用いることができるアジャスタブルプリフェッチ命令キャッシュの構成を示すブロックダイアグラムである。

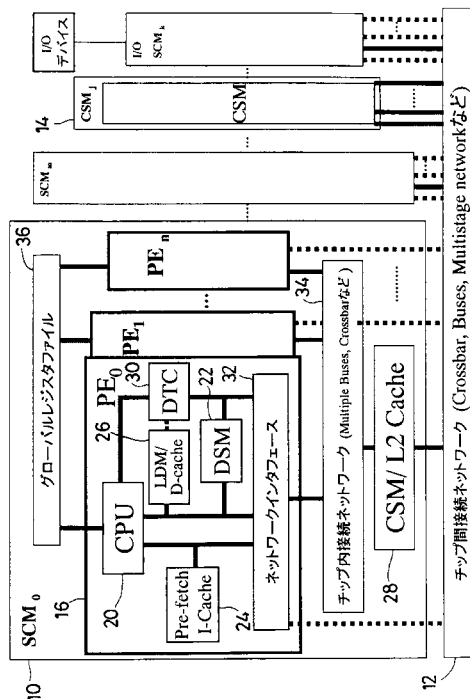
【符号の説明】

- 10 シングルチップマルチプロセッサ
- 12 チップ間接続ネットワーク
- 14 集中共有メモリ(チップ)
- 16 プロセッシングエレメント
- 20 CPU
- 22 分散共有メモリ
- 24 アジャスタブルプリフェッチ命令キャッシュ
- 26 ローカルデータメモリ
- 28 集中共有メモリ
- 30 データ転送コントローラ
- 32 ネットワークインタフェース
- 34 チップ内接続ネットワーク

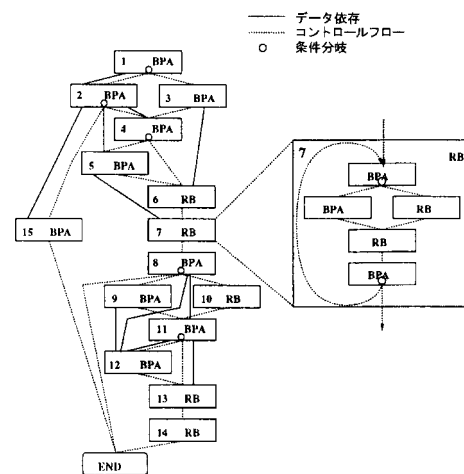
10

20

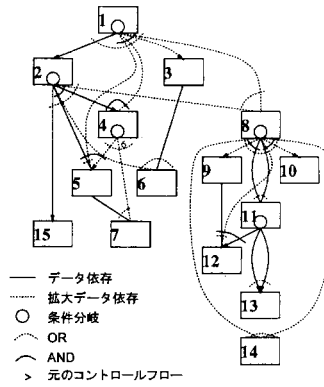
【図1】



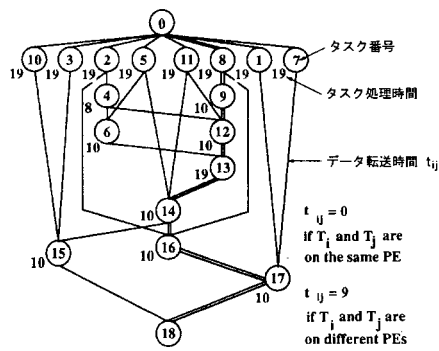
【図2】



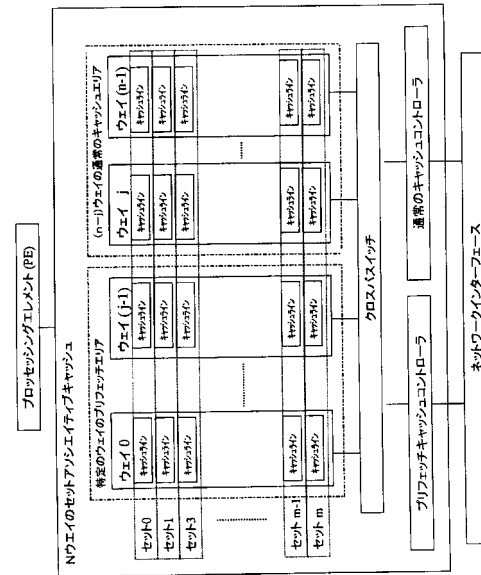
【図 3】



【図 4】



【図 5】



---

フロントページの続き

合議体

審判長 鈴木 匡明

審判官 石井 茂和

審判官 清木 泰